REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-05-

0245

1. REPORT DATE *(DD-MM-YYYY)* 23/6/2005	2. REPORT TYPE Final Report	3. DATES COVERED *(From - To)* 01/7/2004---30/6/2005
4. TITLE AND SUBTITLE Simulation of Quantum Time-Frequency Transform Algorithms		5a. CONTRACT NUMBER
		5b. GRANT NUMBER FA9550-04-1-0406
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Chao Lu Computer & Information Sciences Towson University		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Towson University 8000 York Road Towson, MD 21252		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Jon Sjogren AFOSR/NM 4015 Wilson Blvd, Room 713 Arlington, VA 22203-1954		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The demand for exact computation in scientific fields related to quantum physics is not met by the Symbolic Math Toolbox developed in MATLAB. In particular, exact evaluation rational multiples of 2[illegible]is at the heart of efficient implementation of quantum time-frequency transforms. Computations performed using this Toolbox generate erroneous results when used with numbers with more than twenty digits in length. Furthermore, the results of our investigation lead us to believe that floating-point operations may be used during the computing process of this Toolbox. The *Exact Computing* system introduced in this report yields significant decreases in computation times, as well as providing an exact method of storing and computing data. In this system, exact values are obtained by storing numbers as numerator and denominator of rational numbers. Integers can be of any length.

We define a data structure of rational matrices using rational numbers and a set of related operators. We also started to use alternative approach to represent all integers and rational numbers in terms of a set of residues with respect to a prime number and its powers, called a p-adic number system. The p-adic arithmetic has many attractive features.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Chao Lu
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)* 410-704-3701

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

20050705 051

Simulation of Quantum Time-Frequency Transform Algorithms

Final Report (Grant # FA9550-04-1-0406)

Chao Lu
Computer & Information Sciences, Towson University

June 20, 2005

1. Introduction

In the scientific fields related to quantum physics, one is interested in exact linear computation, and cannot tolerate any round-off/truncation errors introduced by conventional p-ary or floating-point arithmetic when dealing with matrices containing rational entries. The demand for exact computation has led us to explore two possible approaches: (1) rational arithmetic by representing the numerator and denominator of fractional numbers with arbitrary length integers; (2) residue or modulo arithmetic using p-adic number systems [1 Krishnamurthy 77]. The computational complexity of the first approach has been extensively studied. Not only has it been shown that it is very expensive and laborious, but also that it is a very challenging computer science problem to deal with dynamic memory allocation during the computing process. The second approach represents all integers and rational numbers in terms of a set of residues with respect to a prime number and its powers called a p-adic number system. The p-adic arithmetic has many attractive features [1]. Furthermore it appears that an extension of the formalism of quantum theory to the field of p-adic numbers is of great interest even independent of possible physical applications because it may lead to a better understanding of the formalism of quantum theory [2 Vladimirov chapter 3]. We have written C++ routines for approach (1) and utilized part of the NTL and have started to build a computational library using a p-adic system for applications in quantum computing. One specific application of quantum computing, quantum computational Weyl-Heisenberg representations was studied and investigated.

The demand for exact computation in scientific fields related to quantum physics is not met by the Symbolic Math Toolbox developed in MATLAB. In particular, exact evaluation rational multiples of 2π is at the heart of efficient implementation of quantum time-frequency transforms. Computations performed using this Toolbox generate erroneous results when used with numbers with more than twenty digits in length. Furthermore, the results of our investigation lead us to believe that floating-point operations may be used during the computing process of this Toolbox. The *Exact Computing* system introduced in this report yields significant decreases in computation times, as well as providing an exact method of storing and computing data.

In this system, exact values are obtained by storing numbers as numerator and denominator of rational numbers. Integers can be of any length. We define a data structure of rational matrices using rational numbers and a set of related operators.

The *Exact Computing* system is written in Visual C++, which can be called from MATLAB.

We also started to use alternative approach to represent all integers and rational numbers in terms of a set of residues with respect to a prime number and its powers, called a p-adic number system. The p-adic arithmetic has many attractive features [1].

2. Technical Issues Addressed

2.1 Computational Issues of Rational Arithmetic

In rational matrix computation exact values are obtained by storing an arbitrary length rational number as numerator and denominator. We define a data structure of rational matrix operations as a set of operators. The related computational issues are:

a) Data Structure: In order to store data in an exact manner, exact values of rational numbers must be used and maintained throughout the computation.

b) Integers: The system must be capable of storing and manipulating integers of arbitrary length.

c) Dynamic memory allocation: Use of dynamic memory to define rational matrices as well as rational numbers. Properties of the rational matrix to be considered are: size, organizational structure, and implementation of rational matrix calculations.

d) Interface: Once these problems have been addressed, the interface between C++ and MATLAB must be considered if one is interested on calling C++ routines from MATLAB due to arbitrary length.

NTL, written by Victor Shoup, is a high-performance, portable C++ Library for number theory that provides both data structures and algorithms for arbitrary length integers. NTL allows manipulation of integers for vectors, matrices, and polynomials over finite fields, and arbitrary precision floating-point arithmetic.

In our *Exact* Scientific Computational Library, we only use the arbitrary length integer part of NTL to accomplish the task of input, output, and storage of arbitrary length integers.

Further information about the NTL library can be found at: http://shoup.net/ntl/.

We have defined a set of rational number operators as:
Operator +:
```
RationalNumber a, b;
a+b=(a.numerator*b.denominator+b.numerator*a.denominator)
/a.denominator*b.denominator;
```

Operator -:
RationalNumber a, b;
a-b=(a.numerator*b.denominator-b.numerator*a.denominator)
/a.denominator*b.denominator;

Operator *:
RationalNumber a, b;
a*b=a.numerator*b.numerator/(a.denominator*b.denominator);

Operator /:
RationalNumber a,b;
a/b=a.numerator*b.denominator/(a.denominator*b.numerator);

Operator =:
RationalNumber a, b;
a=b means:
a.numertor=b.numerator;
a.denominator=b.denominator;

Simplify ( );
Find the GCD of numerator and denominator; divide both numerator and denominator by the GCD.

Operator ==:
Suppose rational numbers *a and b* are in simplest form. If a.numerator==b.numerator and a.denominator==b.denominator, rational number *a* equals *b*.

These operators serve as the basic functions of the implementation of the rational arithmetic approach. During the computational process all rational entries of a matrix will keep their fractional data type and will not transfer to floating-point. We have implemented some basic rational matrix operations such as: addition, subtraction, multiplication and square matrix inverse. More functions will be added to the library. All routines are written in C++. Interface of our ESCL with MATLAB has been designed and tested. Preliminary results will be reported in section 3.

2.2 Computational Issues of P-adic Arithmetic

The p-adic arithmetic system was introduced for linear computation by Krishnamurthy [1] in 1977. The properties of freedom from round off errors and of simplicity of hardware/software implementation have made it attractive to the scientific community for exact computation. Let us briefly introduce the p-adic number system and its application in related arithmetic algorithms for rational matrix computation.

2.2.1 Segmented P-adic Expansions (Hensel Codes)

Let $\alpha = a/b$ be a nonzero rational number such that $b \neq 0$. Then α can be uniquely expressed as an expansion of powers of a prime p as:

$$\alpha = \sum^{\infty} a_j p_j \qquad (1)$$

Where $0 \le a_i \le (p-1)$. The infinite series (1) converges to the rational number α in the p-adic norm.

The expansion is infinite except in the case of rationals of the form a/p^n for $n \ge 0$.
These rational numbers are called radix fractions.

A general rational number (other than a radix fraction) does not terminate in its p-adic expansion and the convergence to the actual value in the sense that the p-adic norm is obtained only for infinite terms. If we segment the p-adic expansion of a rational number to finite *t* terms, there is a residue equivalent of this finite expansion, Hensel code $H(p,r,\alpha)$, and this expansion form the basis for exact computation [1]. As long as the absolute value of the numerator or denominator does not exceed $\sqrt{(p^r-1)/2}$, the Hensel code is unique.

Let us briefly describe Hensel Codes [1]. If we truncate or segment the *p*-adic expansion of a rational number to a finite number of digits *r*, this truncated number has no resemblance to the rational in the sense of a *p*-adic norm and it corresponds to some radix fraction. However, there is a residue equivalent of this finite expansion and this forms the basis for exact computation. We will discuss this aspect now.

Definition: Let α be a rational number and $a_{-n}, a_{-n+1}, \ldots, a_{-1}, a_0 \ldots a_k \cdots$ be its *p*-adic expansion.

Then the finite segment $a_{-n}, a_{-n+1}, \ldots, a_{-1}, a_0 \ldots a_k \cdots$, where $r = n+k+1$ is called the Hensel code of α and is denoted by $H(p,r,\alpha)$.
For convenience, $H(p,t,\alpha)$ is denoted as an ordered pair in the mantissa-exponent form thus: (m_α, e_α). Since we keep the length of $H(p,r,\alpha)$ constant (*r* digits), e_α is permitted to be zero or to be only negative values. When $e_\alpha = -n$, the radix point is placed *n* digits to the right of the left-most digit of m_α accordingly, the mantissa is an integer and we can always assume that m_α is of the form

$$m_\alpha = a_0, a_1, \ldots, a_{r-1}$$

and

$$e_\alpha \le 0.$$

Let $\alpha = (a/b)p^n$ be a rational number where
$GCD(a,b) = 1$, and $GCD(a,p) = GCD(b,p) = 1$.

Let $H(p,r,\alpha) = (a_0, a_1, \ldots, a_{r-1}, -n) = (m_\alpha, e_\alpha)$ then $m_\alpha = a_0, a_1, \ldots, a_{r-1}$ is the *p*-ary representation of the integer $|a \cdot b^{-1}|_{p^r}$; in other words

$$m_\alpha = \left|a \cdot b^{-1}\right|_{p^r} = \sum_{i=0}^{r-1} a_i p^i .$$

2.2.2 Arithmetic Operations Using Hensel Codes

Basic arithmetic operations using $H(p,r,\alpha)$ codes are essentially modulo p^r arithmetic realized as recursion of modulo p operations. Let $H(p,r,\alpha) = (m_\alpha, e_\alpha)$ and $H(p,r,\beta) = (m_\beta, e_\beta)$ where

$$m_\alpha = \ldots, a_0 \ldots a_{r-1}$$

$$m_\beta = \ldots, b_0 \ldots b_{r-1}.$$

1) *Addition-subtraction*: The algorithm for addition aligns the p-adic point of the mantissa, retaining the lower exponent and finds the sum digit s_i and carry digit c_{i+1} from a knowledge a_i, b_i and c_i.

Thus $s_i = (a_i + b_i + c_i) \bmod p$

for i =0 , 1, 2,..., (r-1)

c_{i+1}=1, if $a_i + b_i + c_i \geq p$

=0, otherwise

$c_0 = 0$, and ignore c_r.

Subtraction is realized as a complemented addition.

2) *Multiplication:* This is similar to p-ary multiplication, except that the product is developed to only lower r digits (modulo p^r) (and hence has a complexity O($r(r$+1)/2)). The algorithm consists in forming the cross-products of the mantissa

$P_{ij} = b_i a_j$ for $0 \leq i \leq (r-1)$

and $j = 0,1,\ldots r-1$, and the partial product P_i and product P thus

$$P_i = \sum_{j=0}^{r-1} P_{ij} \Delta(j)$$

$$P = \sum_{i=0}^{r-1} P_i \Delta(i)$$

where $\Delta(X)$ denotes a right shift of X digits. The exponent of the result is $(e_\alpha + e_\beta)$.

3)*Multiplicative Inverse* and *Division:* Given that $0 \leq m_\beta \leq P^r - 1$ and $GCD(p, m_\beta) = 1, m_\beta^{-1} \bmod p^r$ can be obtained very simply by a recursive solution of the congruence with respect to p.

Let $m_\beta = b_0, b_1, \ldots b_{r-1} (b_0 \neq 0)$ and $m_\beta^{-1} = q_0, q_1, \ldots, q_{r-1}$. The q_i can be obtained by solving for q_i in

$$m_\beta \sum_{i=0}^{r-1} q_i p^i \equiv 1 \bmod p^r.$$

Thus, starting with $q_0 \equiv b_0^{-1} \bmod p, q_k (k \geq 1)$ is computed by solving for

$$(q_k p^k + \sum_{i=0}^{k-1} q_i p^i) b \equiv 1 \bmod p^{k+1}.$$

This leads to the following deterministic trial-error-free division algorithm the quotient, digit by digit, proceeding from the lower index to the higher index position.

The following is the algorithm for finding $m_\alpha \cdot m_\beta^{-1}$.

Let

R_0 zero-th partial remainder or initial numerator (=1 for finding b^{-1});

R_i ith partial remainder;

R_{ii} ith positional digit of R_i.

Then

$$q_i = R_{ii} b_0^{-1} \bmod p$$

for i=0, 1, 2, …, (r-1) and $R_{i+1} = R_i - q_i b \Delta(i)$, where $\Delta(i)$ is the right shift by i digits.

Note that this algorithm can be applied for any numerator; by setting R_{00} = 1 and all other digits of R_0 to zero, one can obtain the multiplicative inverse of m_β. This algorithm has a complexity O(r(r+1)/2). The exponent of the result is $(e_\alpha - e_\beta)$.

2.2.3 Conversion of Hensel Codes to Rationals

Although there are several methods available [5], we will describe here a simple method that is very efficient and economical for matrix computations.

As every rational number a/b ($b \neq 0$. $0 \leq |a| \leq \sqrt{(p^r - 1)/2}$, $0 < |b| \leq \sqrt{(p^r - 1/2)}$) is presented in the form $(a \cdot b^{-1}) \bmod p^r$, it is possible to determine a as well as b if some common multiple of all the denominators involved in a given algorithm is known. Since any algorithm consists of a predetermined sequence of arithmetic operations, it is possible to derive the arithmetic expression for this common multiple, and this facilitates the conversion.

3. Implementations

3.1 Implementation of Rational Arithmetic

We have written some of the routines for this approach and utilized part of the NTL to fulfill the task of input, output and storage of arbitrary length integers. We have defined the rational number data structure with arbitrary length integers, rational operators, rational matrix operations.

For the rational number data structure, the number is stored as its numerator and denominator with arbitrary length integer. This maintains the precision of the number during the calculation. The defined rational operators are +, -, *, / and <, >, <=, >=. During computational process all rational numbers will keep their fractional data type. The floating-point data type was never used, to keep the calculation free of round-off/truncation errors.

We use the following two examples to show the advantages of our programs in terms of computing speed and exactness (correctness) by comparing our results with the results generated by the MATLAB-embedded Symbolic Toolbox.

As far as exactness is concerned, when the input values are small, both methods obtain the same results. However, when the input values reach very large size (20 digits or more), our system can obtain exact results, while the symbolic computation can only get approximations. (We suspect that the Symbolic Toolbox uses floating-point during computations). Our system performs exact computing, while the Symbolic Toolbox cannot be fully trusted in processes requiring exact computation.

Example 1: *When the data sizes are small, the results are the same.*

```
x = [5/7 9/3 0;                          y = [5/7 2/3 0 2/3;
  9/-8 0 3/8;                                 7/8 0 5/3 6/5;
  -2/3 2/3 3/4;                               2/3 2/5 3/4 7/8];
  5/6 7/-8 0];
```

```
ESCL_multiplyResult_1 =
  '1229/392'    '10/21'    '5'          '428/105'
  '-31/56'      '-3/5'     '9/32'       '-27/64'
  '17/28'       '-13/90'   '241/144'    '1457/1440'
  '-229/1344'   '5/9'      '-35/24'     '-89/180'
```

```
symbolicMultiplyResult_1 =
[  1229/392,     10/21,          5,   428/105]
[    -31/56,      -3/5,       9/32,    -27/64]
[     17/28,    -13/90,   241/144, 1457/1440]
[ -229/1344,       5/9,     -35/24,   -89/180]
```

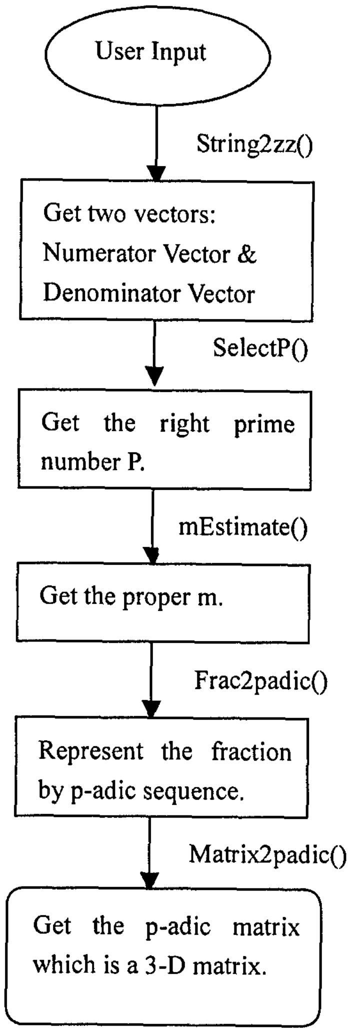
Example 2: *When the data sizes are large, the results are different. The special data set chosen can easily show that our result is correct, while the Symbolic Toolbox is not.*

```
x=[123456789987654321/77777777777777777777,     88888888888888888888/33 ;
  -123456789987654321,      88888888888888888888/33];

y=[77777777777777777777/123456789987654321,    1   ;
  33/88888888888888888888,            0];

ESCL_MultiplyResult_2 =
[2                             111111111/70000000007]
[-77777777777777777776   -123456789987654321   ]

symbolicMultiplyResult_2 =

[81129638414606679562133097328419/405648192073033408478945025720 32,
183003413228354 5/1152921504606846976]
[-315504149390137066134002210479948803686439144719630 1/405648192073033408478945025720
32,   -123456789987654320]
```

To demonstrate the gain in computational speed, the reports generated by MATLAB profiles show that the results of matrix multiplication with small rational entries, our system is about 2.5 times faster, and square matrix inverse is about 5 times faster than the MATLAB results. We only compared two systems of matrices with small rational entries, since MATLAB can only give correct results in this case.

3.2 Implementation of Matrix Computation Using P-adic Expansions

Program Overview Flowchart:


User Input
String2zz()
Get two vectors: Numerator Vector & Denominator Vector
SelectP()
Get the right prime number P.
mEstimate()
Get the proper m.
Frac2padic()
Represent the fraction by p-adic sequence.
Matrix2padic()
Get the p-adic matrix which is a 3-D matrix.

Detail explanation of the program:

User Input

FileName: ZZTools.cpp.

It includes these two functions:

void string2zz(string ell, int roww, int coll,vec_ZZ &allnum,vec_ZZ &allden) ,

ZZ toInteger(string str,int len).

Input: Rational matrix (the length of each elem mEstimate() be arbitrary long)

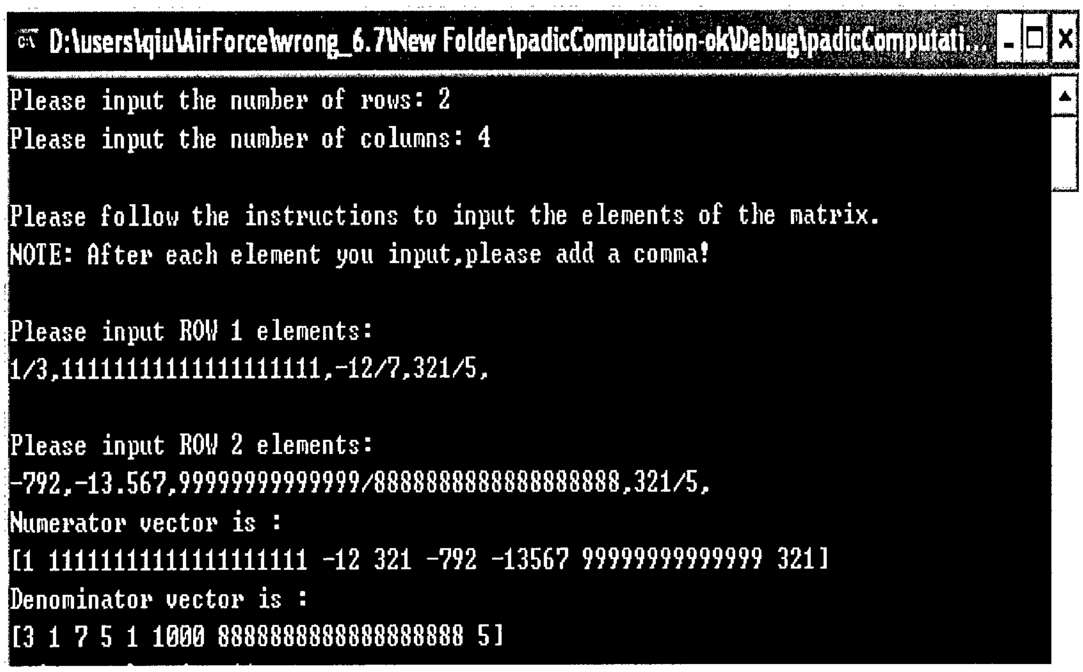
Output: Numerator vector and Denominator vector.

It is known that the normal data structures cannot deal with arbitrary length integer due to overflow if the input number is too long. The NTL library is used to handle this problem in our program.

The following is the description of user interface:

First, the program reminds the user to input the number of rows and columns of a matrix.
Second, the program asks the user to input the element in first row, second row and so on.
Third, the program saves input into two ZZ type vectors; both of them have the same size, rows-by-columns. One is used to save numerators and the other one is used to save denominators. The two vectors will be shown on the screen.

Demo of input process: We can see the user interface from the following diagram. It shows that this process can deal with all kinds of input successfully, such as arbitrary long integers (1111111111111111111111111...), fractions (1/3), negative numbers (-792, -12/7) and decimals (12.5, -13.567).


D:\users\qiu\AirForce\wrong_6.7\New Folder\padicComputation-ok\Debug\padicComputati...

```
Please input the number of rows: 2
Please input the number of columns: 4

Please follow the instructions to input the elements of the matrix.
NOTE: After each element you input,please add a comma!

Please input ROW 1 elements:
1/3,11111111111111111111,-12/7,321/5,

Please input ROW 2 elements:
-792,-13.567,99999999999999/8888888888888888888,321/5,
Numerator vector is :
[1 11111111111111111111 -12 321 -792 -13567 99999999999999 321]
Denominator vector is :
[3 1 7 5 1 1000 8888888888888888888 5]
```

Prime number selection:
FileName: selectP.cpp
It includes the following two functions:
vec_ZZ listPrime(),
ZZ selectP(vec_ZZ vecDen).
Input: Denominator vector.
Output: Smallest Proper prime number.

In C++ program, we implement this process as:
First, we generate the prime number in order.
Second, select the prime number that cannot divide each element of the denominator vector.
Example: for a given rational matrix:

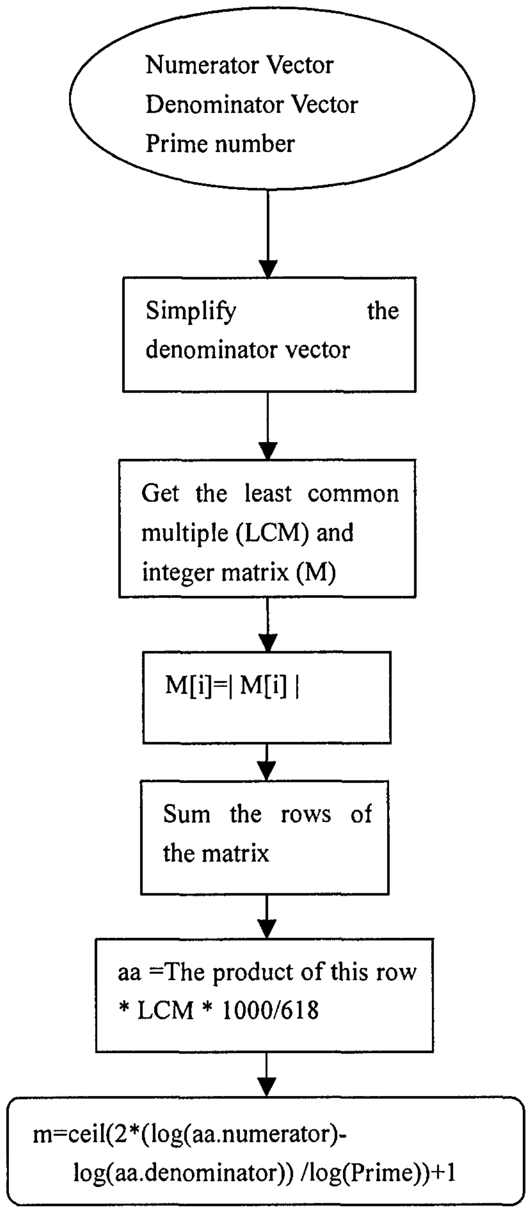
[2/5, 0.6
2/3, 0.75],
the smallest prime number is 7.

Find the proper number of digits of a p-adic sequence for the whole matrix

FileName: PredictM.cpp.
It includes the following functions:
void Demon(vec_ZZ inputNum, vec_ZZ inputDen,vec_ZZ& outNum,vec_ZZ& outDen, ZZ& Gc);
void Gcd0(ZZ f,ZZ g, ZZ& del, ZZ& gam, ZZ& k, ZZ& el);
void sumRows(vec_ZZ inputNum, vec_ZZ inputDen,vec_ZZ& rowNum, vec_ZZ& rowDen, long m, long n);
long mEstimate(vec_ZZ inputNum,vec_ZZ inputDen, ZZ prime,long m,long n);
Input: Numerator Vector, Denominator Vector, prime number and the size of matrix.

Output: Proper *m* (*m* is the number of digits of the p-adic sequence).
The given numerator vector and denominator vector have been simplified.

If it is a fractional number, for example 0.618, we take numerator = 618 and denominator = 1000.
The following diagram shows this process:


Numerator Vector
Denominator Vector
Prime number
Simplify the denominator vector
Get the least common multiple (LCM) and integer matrix (M)
M[i]=| M[i] |
Sum the rows of the matrix
aa =The product of this row * LCM * 1000/618
m=ceil(2*(log(aa.numerator)-log(aa.denominator)) /log(Prime))+1

Example:
Input:
numerator vector is: [2 3 2 3];
denominator vector is : [5 5 3 4];
prime number is : 7

After simplifying, the denominator vector is : [5 1 3 4];

The least common multiple LCM=60, and integer matrix is:
[24, 36
40, 45].

Sum the rows of the matrix, then we can get a row vector:
[64 81]

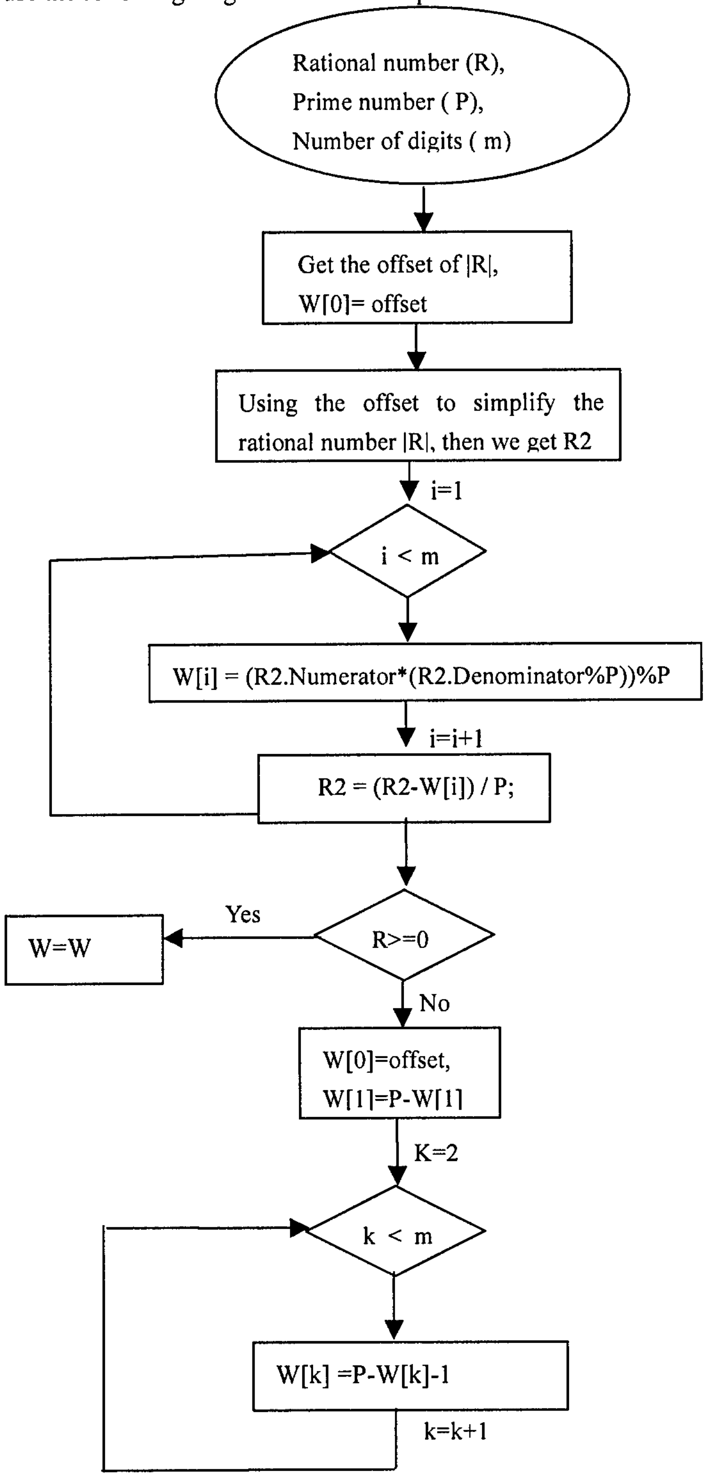
aa.numerator is: 51840000
aa.denominator is: 103

The result m=15

Expand the single fractional number into a P-adic sequence

FileName: frac_Padic.cpp
It includes this function:
void padicExpand(vec_ZZ& w, ZZ prime, long numdig),
and it calls the following functions:
ZZ BoundP(ZZ b , ZZ prime);
ZZ invertp(ZZ n, ZZ prime);
ZZ powmod(ZZ v ,ZZ t, ZZ prime);
RationalNumber* Simplify();
ZZ powerP(ZZ p, ZZ t).

Input: the numerator and denominator of a rational number, prime number, digits number of the p-adic sequence.
Output: P-adic sequence vector for the rational number.
We use the following diagram to show this process:

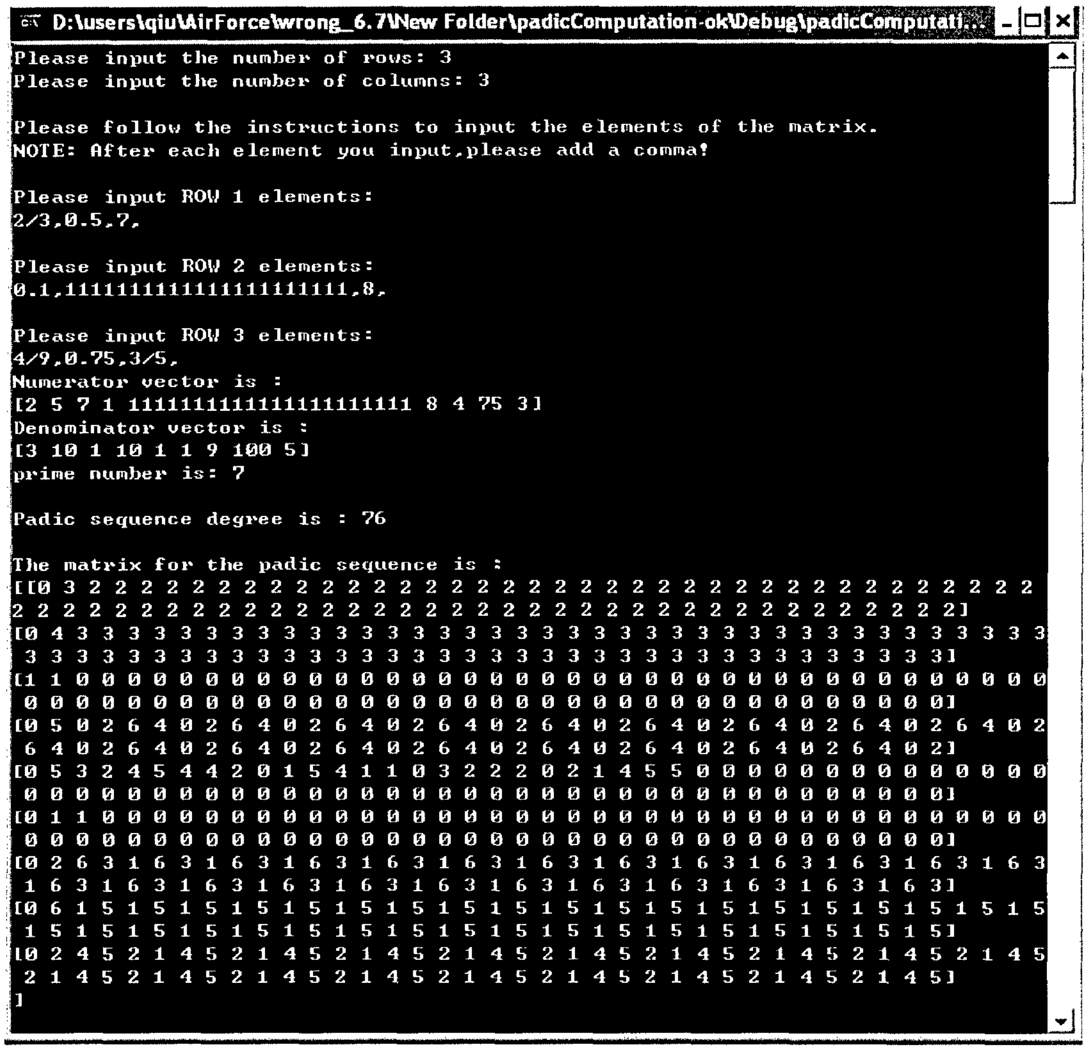


Example:
Input: R= -7/9, P=2, m=15,
Output: [0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0].

Represent a rational matrix using p-adic expansion matrix

FileName: matrixCal.cpp
It includes the function: ***void matrix2padic***(vec_ZZ vecNum,vec_ZZ vecDen,mat_ZZ& matPadic,long numdig,ZZ prime).
Input: Numerator vector, denominator vector, prime number, and the number of digits.
Output: P-adic sequence matrix.

The implementation: For each element of the rational matrix, we call ***padicExpand(..)*** function in the file "frac_Padic.cpp" , then we can get a p-adic sequence vector for this rational number. Repeat for all the elements of the matrix, finally, we get the p-adic sequence matrix for the rational matrix.
The following chart is an example.



```
D:\users\qiu\AirForce\wrong_6.7\New Folder\padicComputation-ok\Debug\padicComputati...
Please input the number of rows: 3
Please input the number of columns: 3

Please follow the instructions to input the elements of the matrix.
NOTE: After each element you input,please add a comma!

Please input ROW 1 elements:
2/3,0.5,7,

Please input ROW 2 elements:
0.1,1111111111111111111111,8,

Please input ROW 3 elements:
4/9,0.75,3/5,
Numerator vector is :
[2 5 7 1 1111111111111111111111 8 4 75 3]
Denominator vector is :
[3 10 1 10 1 1 9 100 5]
prime number is: 7

Padic sequence degree is : 76

The matrix for the padic sequence is :
[[0 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
[0 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
[1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 5 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2
 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2 6 4 0 2]
[0 5 3 2 4 5 4 4 2 0 1 5 4 1 1 0 3 2 2 2 0 2 1 4 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 2 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3
 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3 1 6 3]
[0 6 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5
 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5]
[0 2 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5
 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5 2 1 4 5]
]
```

Future work

In the future, we are going to do the following:

1) Operations in the p-adic sequence domain, such as addition, subtraction, multiplication, division, matrix inverse and so on.
2) Conversion from p-adic expansion domain to fractional domain.
3) Design friendly user interface.

4. References

[1] Krishnamurthy, F. V. "Matrix Processors Using P-adic Arithmetic for Exact Linear Computations", IEEE Transactions on Computers, vol. C-26, No. 7, July 1977.

[2] Vladimirov, V.S., Volovich, I.V. and Zelenov, E.I. *P-adic Analysis and Mathematical Physics,* Series on Soviet & East European Mathematics – Vol. 1.World Scientific 1994.

[3] Lu, C. and An, M. "Progress Report", 2004.

[4] Shoup, V *NTL library* at: http://shoup.net/ntl/

[5] Kornerup, P. and Gregory, R.T. "Mapping Integers and Hensel Codes onto Farey Fractions", BIT 23 (1983), 9-20.

[6] Dixon, J. "Exact Solution of Linear Equations Using P-adic Expansions", Numerische Mathematik 40, 137-141 (1982) Springer-Verlag.